Proceedings of International Conference on Distributed
Systems, Software Engineering and Database Systems

Evidence-based Software Verification and Validation Technique*

Chin-Feng Fan, Wei-Huo Huang
Dept. of Computer Engineering and Science
Yuan-Ze Institute of Technology, Chung-Li, R.O.C.

Swu Yih
Division of Nuclear Instrumentation and Control
Nuclear Energy Research, Lung-Tang, R.O.C.

Abstract

Current software verification and validation (V&V) practice lacks a systematic and integrated methodology; moreover, the quality of V&V process is difficult to judge. This paper proposes an Evidence-Based software Verification and Validation (EB V&V) technique to solve these problems. In the paper, V&V process is viewed as an evidence searching process; we first propose an evidence taxonomy, which divides evidence into syntactic, semantic, complete, and exception impact (also called safety) types of evidence. Methods to obtain these types of evidence along with evidence formats have been discussed. A quality-chart approach is then developed to evaluate the quality of V&V process as well as that of the evaluated software. An evidence database can be built to store and organize the obtained evidence for further utilization. A case study is also presented.

1. Introduction

Verification and validation is a major technique to improve software quality. For safety critical software, which used in aviation, transportation, medicine, and nuclear industry, V&V is particularly important; V&V results can be used in licensing approval process and in liability determination after an accident has occurred. Current software V&V practice lacks a systematic approach; it usually generates many independent reports, which are not integrated and cannot be fully comprehended and utilized. V&V can be viewed as an *evidence*-searching process, searching for the connection between contiguous development stages and between software and the requirements. V&V activities include testing and review. Testing cannot cover all possibilities; hence, review process is important. However, review is a mental activity; most of the evidence obtained by searching and comparison during review process is in the reviewer's mind and is not recorded. Therefore, it is difficulty to judge the quality of the review process. To solve these problems, we propose an Evidence-Based software Verification and Validation approach (EB V&V), which first derives a set of evidence that reflects the essential relationship between software development phases, and then, explicitly stores the evidence in an integrated database. In this approach, an evidence taxonomy is proposed to systematically classify evidence; techniques for evidence acquisition, representation, and integration in a database are developed. A quantitative approach to estimate V&V quality is also proposed. The EB V&V methodology is presented in Section 2, followed by a case study in Section 3 and a conclusion in Section 4.

2. Evidence-based Software V&V Technique (EB V&V)

2.1. The Rationale

Under the waterfall software development life cycle model, the development activities are divided into several consecutive phases as shown in Figure 1. The developers in each phase have the same goal to achieve and the same target to construct; since developers in each phase use different languages, thus, the development process can be viewed as a

This work was supported in part by National Science Council, Republic of China, under the grant no. NSC85-2213-E-155-004.

translation or refinement process (if with the same language) as shown in Figure 2. Under this idea, the output of one phase is the subset or refinement of the design of its previous phase. The output of each phase can be viewed as a set of expressions as follows:

User's requirements: $U=(U_1, U_2, .., U_m)$

It is a collection of natural language description orgnized in itemized forms U_i 's.

Requirements specifications: $R=(R_1,R_2,..R_n)$

Each user requirement item is translated into one or multiple specifications R_i's.

Design phase: $D=(D_1,D_2,..D_p)$

Each requirment specification is translated into one or multiple deisgn entities D_i's.

And so on.

Each expression represents a description of software state transition based on the language used in that phase. For example,

$$R_i = State_i^R \xrightarrow{input_i^R} State_j^R$$

refers to a particular state transition perceived in the requirement phase for requirement item i and represented in the requirement specification lagnuage. There exists connection between this required transition with its implemnetation, which may be implemented by a set of transitions, in the next stage. Thus, the evidence in V&V process will link such connection. For example, the evidence V_i associates the state transition in the requirment phase from state i to j with its implementation in the design phase; such connection will be denoted as

$$< R_i, D_i^*, V_i >$$

where,

$$R_i = State_i^R \xrightarrow{input_i^R} State_j^R$$

$$D_i^* = P(State_i^D) \xrightarrow{P(input_i^D)^*} P(State_j^D)$$

V_i = Evidence

Note that

P refers to power set, and

$State_i^R \supseteq P(State_i^D)$,

$State_j^R \supseteq P(State_j^D)$,

$input_i^R \supseteq P(input_i^D)$

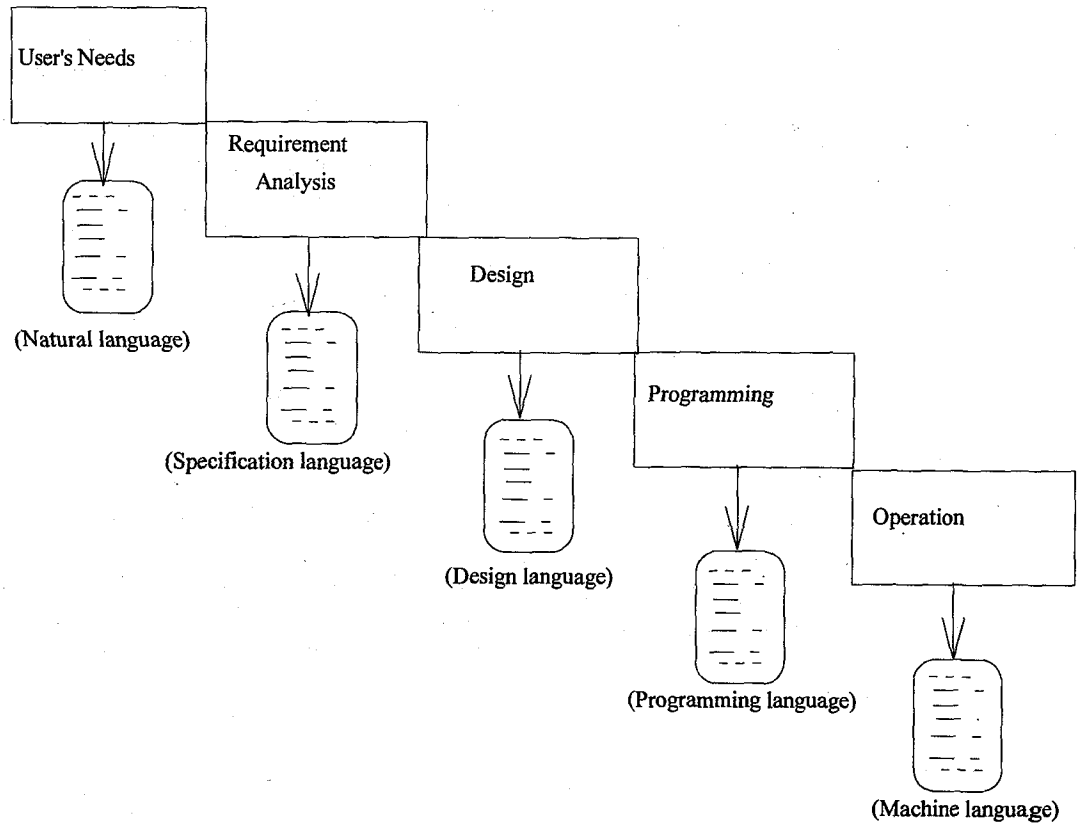


Figure 1. Language used in different software development phases

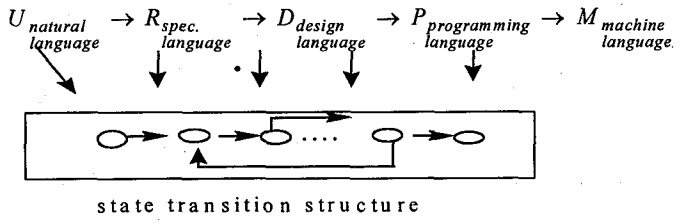


Figure 2. Software development as a translation or refinement process

The related sets of state transitions are the basic unit in the evidence-based approach; this basic unit is named as *entity*. Thus, evidence in EB V&V approach connects a particular entity in a development phase with its refinement or implementation entities in the next development phase. For each phase, the entity will be different; for example :

User requirement phase: an entity is a requirement item or keywords.

Requirements specifications phase: an entity is a requirement specification item.

Design phase: an entity is a design unit.

Implement phase: an entity is a module in the program code.

Test phase: an entity is a test case.

2.2. Evidence Taxonomy

To facilitate systematic acquisition and utilization of the evidence, we classify the evidence into four types: syntactic evidence, semantic evidence, complete evidence, and exception impact evidence (safety evidence).

Syntactic : The syntactic evidence detects the lexical existence of a given entity.

Semantic: The semantic evidence determines whether a given entity performs its intended objectives.

Complete: The complete evidence determines whether the software completes the exact specification of the previous phase and no unintended extra functions.

Exception impact (Safety): Since completeness is difficult to establish, thus, the evidence is needed to identify the impacts due to incomplete coverage. Such evidence is named as *safety evidence*, detecting if there exist any unsafe factors in the software.

For a particular entity, V&V process will seek the evidence of its refinement or implementation in the next stage. First, the syntactical evidence is sought to show the existence of such refinement or implementation in the next stage; then the semantic evidence is sought to judge the correctness of such implementation. For example, for a requirement specification item "interrupt", the syntactical evidence in the design verification is that the term occurs in the design document; the semantic evidence is that the design properly implements this function. After that, the overall complete evidence can be searched for to demonstrate that the software implemented in the next stage fulfills the specifications of the previous stage and has no unintended functions. However, since the completeness cannot be ensured or may not exist, safety evidence is needed to investigate the impacts induced by potential incompleteness. Safety evidence is particularly important for safety-critical applications: as long as the incomplete portions will not cause unsafe consequences, then the implementation will be acceptable. Syntactic, semantic, and complete types of evidence are sought sequentially; while safety evidence obtained from software failure modes and effect analyses seeks the safety links between stages. Complete and safety evidence mainly refer to the entire software; yet, they can also refer to the complete and safe implementation of a particular entity, if appropriate, by the next stage.

2.3. Evidence Processing

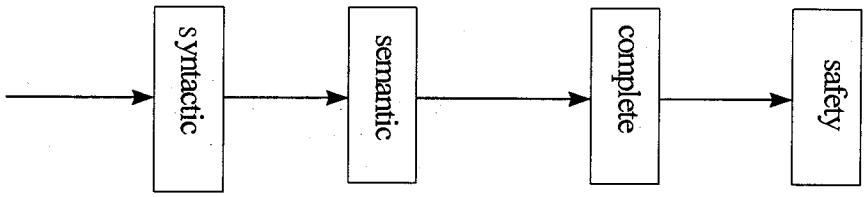
Besides subjective judgment, the above four types of evidence can be obtained by more objective techniques. Syntactic evidence can be identified by string/token mapping. Semantic evidence can be generated by program testing (for implementation stage) or prototype testing (for requirement and design stage). Complete evidence can be processed in several aspects: first, check gathered semantic evidence to verify that all the defined entities in a particular phase have been successfully implemented by the next development phase; then, check using checklists or inspection to determine whether there is no extra/unwanted functions from this implementation by using checklists; also, reliability data can be used to indicate the degree of completeness. A formal

or a semi-formal approach may also be applied for semantic and complete evidence [3,4]. Safety evidence in each development phase can be generated by safety analysis techniques such as accident scenario tree analysis [8] at requirement stage, Petri-net analysis [5] and frame-based fault-tree/event tree analysis [2] at design stages, and software fault tree analysis[6] at implementation stage. These safety analysis techniques will identify the hazardous causes for each development phase, if these causes have been considered and prevented in the current development, then we indicate the places they are treated; otherwise, the safety evidence is missing and needs to be added into development. Figure 3 is a summary of the evidence acquisition process.

The above four types of evidence can be represented in tracing tables. These tracing table will be kept in an integrated evidence database to facilitate further inquiry and utilization. The concept of evidence database is shown in Figure 4. A relational data model may be used to implement this evidence base.

2.4. Quality Index of EB V&V

V&V tasks consists of review and testing. Conventional review process is an internal and mental activity; the EB V&V approach converts this opaque, mental activity to an external and observable process. Consequently, quantitative measurement of EB V&V process is then possible. A quantitative quality chart is proposed to measure the quality of EB V&V tasks in this evidence approach. A quality chart as indicated in Figure 5 is proposed to indicate the percentage of evidence searched out of the all the desirable evidence.



syntactic: token/string mapping

semenatic : program testing,
prototype testing,
or formal approach

complete: 1. check coverage of semantic evidence
for defined entities
2. check no extra functions produced by
checklist or subjective judgment
3. use reliability data as completeness indicator
or formal approach

safety: safety analysis techniques
at each phase

Figure 3. Evidence Acquisition

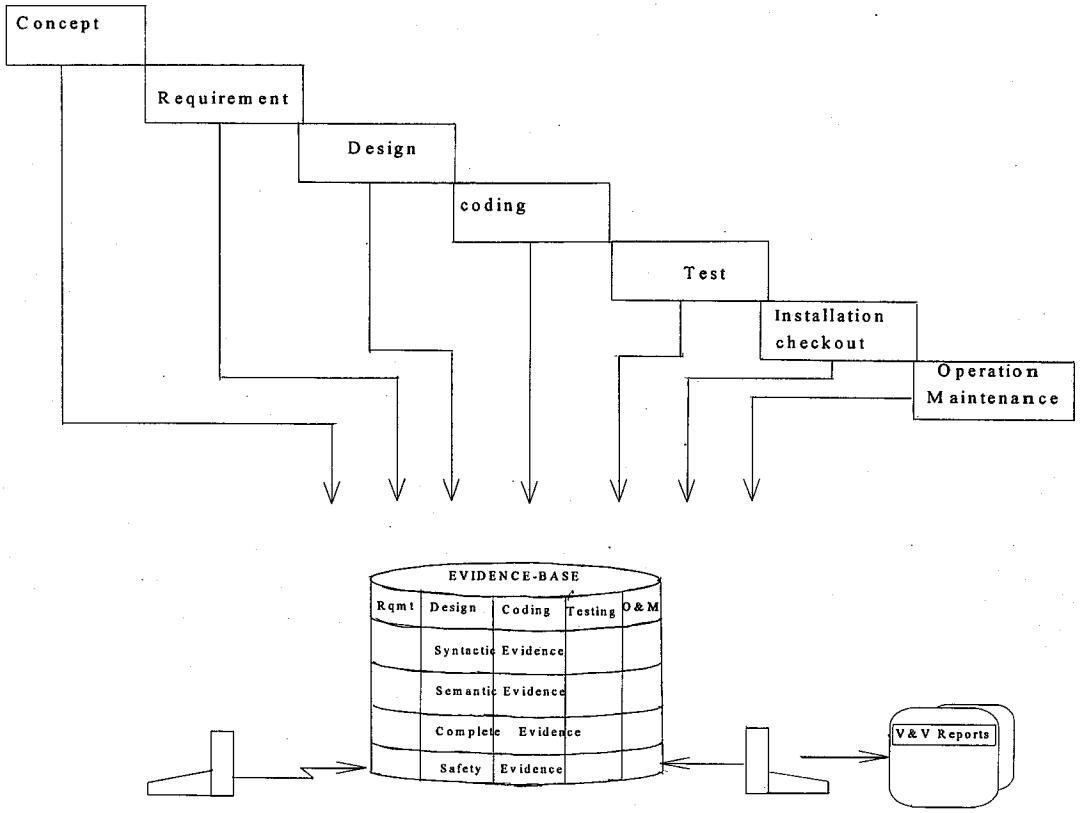


Figure 4. The concept of evidence-base

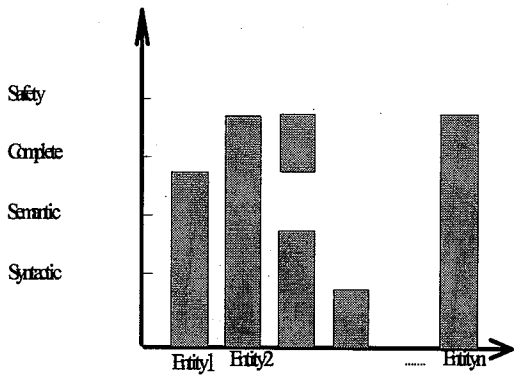


Figure 5. Quality chart

One possible quality metrics of V&V task is as follows:

$$\text{Quality(V\&V)} = \left(\sum_{k=1}^{n} \sum_{T=1}^{4} W_T \bullet E_k(T) \bullet W_{E_k} \Big/ \sum_{k=1}^{n} \sum_{T=1}^{4} W_T \bullet W_{E_k} \right) * 100\%$$

where

T=1..4, represents the four types of evidence.

W_T: weights for types T evidence (W_1,W_2, W_3,W_4)

E_k: entity k. Assume there are n entities at the previous stage.

$E_k(T)$: effort to search T type of evidence for entity K. $E_k(T)$ =0 means the verifier has not tried to examine the evidence, $E_k(T)$=1 means he has.

W_k: weight of entity K.

A further refinement by using error seeds to obtain more accurate V&V quality [1] is desirable. Also, similar quality chart approach can be applied to get *software quality* reflected by the obtained evidence, assuming that the V&V task has high quality first. In such a case, then $E_k(T)$ in the above formula refers to whether such evidence exists.

3. Case Study

We have constructed a simulator for a rapid railroad system to demonstrate the usage and effectiveness of the proposed approach. The simulated system has eight stations (A,B,C,D,E,F,G,H) and four trains (1,2,3,4) staying at stations A,C,E,G at the beginning; this is shown in Figure 6.

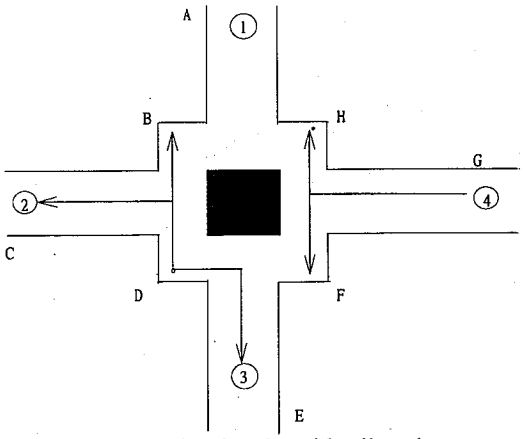


Figure 6 A simulated rapid railroad system

User's requirements includes the following:

1. Trains can move either clockwise or counterclockwise.
2. Trains stop at any station and passengers can get in or leave from the trains.
3. The speed of each train can be changed.
4. The trains follow traffic lights.
5. This system takes speed, number of passengers, and direction as input; either from users, or randomly generated by the computer.
6. The system can be interrupted by the user at any time.
7. The system must has the self-test function.
8. The system can be restarted if needed.
9. The user can check the current state.
10. The system should print out data when necessary.

The documents at the requirements specifications, design, and testing (including prototype testing) stages were written in SDG 2.0 [7]. The requirements specifications identified functional items, including the user inputs the direction of each train, the computer randomly generates the direction of each train, interrupt function, restart function, self-testing, etc. The design stage developed several design modules including window_control, car_control, signal_control, calculate, flow_control, etc. The simulated system was coded in Visual Basic.

The V&V process was proceeded as that discussed above. Tables 1 to 3 show segments of some tracing tables generated by the EB V&V for this case.

Table 1. Syntactical evidence for requirement verification

User's requirements			**Requirements Specifications in SDG2.0**		
No.	**Page**	**Entity**	**No.**	**Page**	**Corresponding Entity**
6	1	The system can be interrupted by the user at anytime.	29	3-20	3.2.11 Item 11: Interrupt function
			30	3-21	3.2.12 Item 12: Restart function
		.....			.....

Table 2. Semantic evidence for requirement verification

User's requirements			**Prototype Testing Report for Specifications**		
No.	**Page**	**Entity**	**No.**	**Page**	**Corresponding Entity**
1	1	Trains can move either clockwise or counter-clockwise.	1	1	Prototype Test case 1
7	1	The system must has the self-test function.	26	25	Prototype Test case 26
		....			...

Table 3 Compete evidence by checklist for requirements

Checklist Item	Rating	Comment
Do the requirements specifications include all functions called for or implied by the Statement of Problem?	Perfect Good *OK* Poor	
Does the specification reference all desired development standards?	Perfect *Good* OK Poor	
Does the requirements consider incorrect input handling ?	Perfect Good OK *Poor*	add negative input...
.....	.....	....

Table 4. Safety evidence for requirement verification

safety analysis document			requirement document in SDG 2.0		
No.	Page	Entity	No.	Page	Corresponding Entity
1	1	User input checking			missing
2	1	run-time self checking	26	2-1	2.2. self-check function
3	1	interrupt function	29	3-20	3.2.11 interrupt function
			30	3-21	3.2.12 restart function

For safety evidence, appropriate analysis techniques needs to be used. For example, the accident scenario tree [8] or fault tree analysis can be performed in the requirements verification to identify that the train collision accident is caused by the following potential reasons:

(1)incorrect user inputs,
(2)lack of run-time self-check, or
(3)lack of interrupt function.

Thus, *negation* of these reasons constitute the safety constraints for requirements specifications. These constraints needs to be checked against the requirements specifications to verify their existence or missing, and the information will be recorded in a safety evidence tracing table, as shown below in Table 4.

We have implemented the gathered evidence in an evidence base using Visual Foxpro in Chinese. The user indicates the entity, the current stage, and the evidence type to search for, the system will show the evidence linked to the previous and the next development stages; this is shown in Figure 7. For detailed descriptions for each searched item, the user can click the item twice, and the detailed information will be linked to (see Figure 8). Since evidence is kept on-line and can be tracked backwards and forwards, further utilization of this information is possible.

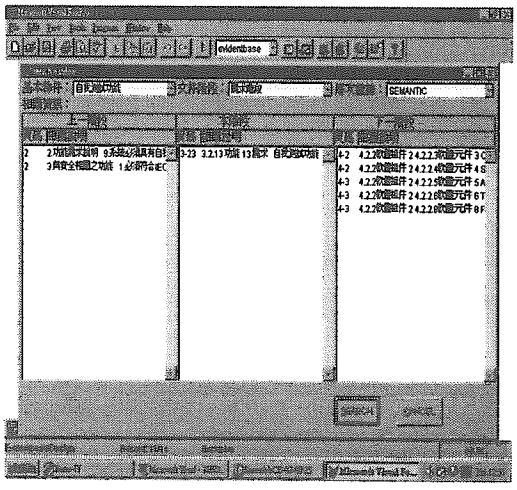


Figure 7. The main screen of the evidence base

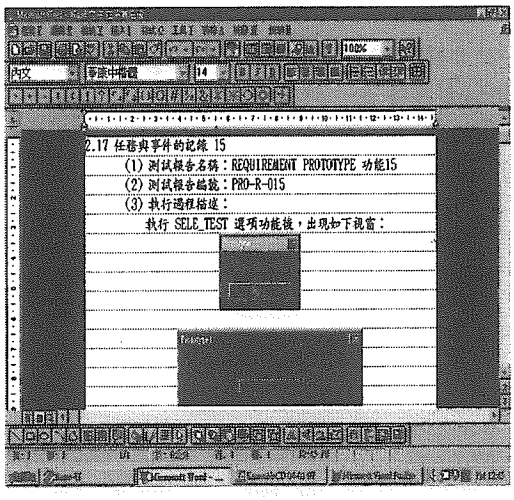


Figure 8: Linked information for a query

4. Conclusion

This paper proposed an evidence-based software verification and validation method, which has the following advantages over the current practice:

1. It is an integrated and systematically V&V method with proposed evidence taxonomy, acquisition procedure, and representation formats.
2. This technique converts the originally internal, mental V&V activities to external, observable behavior by recording detailed evidence.
3. Quality indices measuring V&V performance and results are proposed.
4. Obtained evidence is kept on-line in an evidence base, supporting forward and backward traceability. Thus, V&V information can be further utilized, such as in certification and liability determination for safety-critical software.

Techniques and tools to facilitate the evidence creation and maintenance will be further developed to improve the efficiency of this approach.

References

[1] C. Fan, and S. Yih, "Prescriptive metrics for quality assurance," in Proc. of IEEE Asian Pacific Software Engineering Conf.(APSEC'94) , Tokyo , Dec 1994 , pp. 430-438.

[2] C. Fan and S. Yih, "Frame-based safety analysis approach for decision-based errors," to appear in *Reliability Engineering and System Safety*, 1996.

[3] M. Heimdahl, and N. Leveson, "Completeness and consistency analysis of state-based requirements," in Proc. of 17th International Conference on Software Engineering, April 1995, pp. 3-14.

[4] C. Heitmeyer, B. Labaw, and D. Kiskis, "Consistency checking of SCR-style requirements specifications," in Proc. of International Symposium on Requirements Engineering, March, 1995.

[5] N. Leveson and J. Stolzy, "Safety Analysis Using Petri Nets," *IEEE Trans. On Software Engineering*, SE-13, No. 3, 1987, pp. 386-397.

[6] N. Leveson, S. Cha, and T. Shimeall, "Safety verification of ADA programs using software fault trees," *IEEE Software*, July 1991, pp. 48-59.

[7] Software Development Guide (SDG) 2.0, Information Industry Institute, Dec. 1988, Taiwan

[8] S. Yih and C. Fan, "Development and verification of licensable requirements specifications for safety-critical software," 9th Sino-Japanese Nuclear Safety Conference, Taipei, Dec. 1994, pp. I-F-1-I-F-14.